

# DBMS Lab Experiment 11

## To understand the concepts of index

- **Course:** [DBMS](#)
- **Name:** Devesh Chandra Srivastava
- **SapID:** 590017127
- **Batch:** 66
- **Semester:** 3
- **Date:** 2025-09-08

1. Create an index of name `employee_idx` on `EMPLOYEES` with column `Last_Name`, `Department_id`

```
CREATE INDEX employee_idx ON EMPLOYEES (Last_Name, Department_id);
```

2. Find the `ROWID` for the above table and create a unique index on `employee_id` Column of the `EMPLOYEES`.

```
SELECT ctid, Employee_id FROM EMPLOYEES;
```

```
CREATE UNIQUE INDEX emp_unique_idx ON EMPLOYEES (Employee_id);
```

## Result

### Query Result 1

ctid	employee_id	first_name	last_name	dob	salary	department_id
(0,1)	E001	Amit	Sharma	"1990-05-12T00:00:00.000Z"	55000.00	D01
(0,2)	E002	Riya	Verma	"1992-03-23T00:00:00.000Z"	60000.00	D02
(0,3)	E003	Rahul	Patel	"1988-11-02T00:00:00.000Z"	75000.00	D01
(0,4)	E004	Sneha	Reddy	"1995-09-19T00:00:00.000Z"	50000.00	D03
(0,5)	E005	Karan	Mehta	"1991-12-11T00:00:00.000Z"	68000.00	D02
(0,6)	E006	Neha	Singh	"1993-07-30T00:00:00.000Z"	72000.00	D03

3. Create a reverse index on `employee_id` column of the `EMPLOYEES` .

```
CREATE INDEX emp_reverse_idx ON EMPLOYEES (REVERSE(Employee_id));
```

4. Create a unique and composite index on `employee_id` and check whether there is duplicity of tuples or not.

```
CREATE UNIQUE INDEX emp_composite_idx ON EMPLOYEES (Employee_id,  
Department_id);
```

```
SELECT Employee_id, Department_id, COUNT(*)  
FROM EMPLOYEES  
GROUP BY Employee_id, Department_id  
HAVING COUNT(*) > 1;
```

5. Create Function-based indexes defined on the SQL functions `UPPER ( column_name )` or `LOWER ( column_name )` to facilitate case-insensitive Searches (on column `Last_Name` ).

```
CREATE INDEX emp_lastname_upper_idx ON EMPLOYEES (UPPER>Last_Name));
```

6. Drop the function based index on column `Last_Name` .

```
DROP INDEX emp_lastname_upper_idx;
```

## Comparison between indexing and non indexing

1. Query to insert 10,000 values

```
CREATE TABLE employees (  
    employee_id    SERIAL PRIMARY KEY,  
    first_name     VARCHAR(30) NOT NULL,  
    last_name      VARCHAR(30) NOT NULL,  
    dob            DATE,  
    salary         NUMERIC(10, 2),  
    department_id  INT  
);
```

```
INSERT INTO employees (first_name, last_name, dob, salary, department_id)  
SELECT  
    'First_' || g,
```

```
'Last_' || (g % 500),
DATE '1980-01-01' + (g % 1000),
(30000 + (random() * 50000))::NUMERIC(10,2),
(1 + (g % 10))
FROM generate_series(1, 10000) AS g;
```

## 2. Analysing without indexing

```
EXPLAIN ANALYZE SELECT * FROM employees WHERE last_name = 'Last_250';
```

## Result

### Query Result 1

#### QUERY PLAN

Seq Scan on employees (cost=100000000000.00..10000000138.65 rows=18 width=184) (actual time=0.200..1.600 rows=20 loops=1)

Filter: ((last\_name)::text = 'Last\_250'::text)

Rows Removed by Filter: 9980

Planning Time: 0.200 ms

Execution Time: 2.000 ms

## 3. Create Index

```
CREATE INDEX emp_lastname_idx ON employees (last_name);
```

## Results

### Query Result 1

#### QUERY PLAN

Bitmap Heap Scan on employees (cost=4.67..87.02 rows=50 width=184) (actual time=0.000..0.100 rows=20 loops=1)

Recheck Cond: ((last\_name)::text = 'Last\_250'::text)

Heap Blocks: exact=20

-> Bitmap Index Scan on emp\_lastname\_idx (cost=0.00..4.66 rows=50 width=0) (actual time=0.000..0.000 rows=20 loops=1)

Index Cond: ((last\_name)::text = 'Last\_250'::text)

Planning Time: 0.400 ms

Execution Time: 0.100 ms

## Comparison

Test Case	Planning Time	Execution Time
Without Index	0.200 ms	<b>2.000 ms</b>
With Index	0.400 ms	<b>0.100 ms</b>